P4-单周期CPU电路（2）

0.ISE使用指南

- ise如何确立文件之间的父子关系？

  当你完成子模块的实例化后，软件会自动确立。

- ise仿真时间修改
  - 直接操作：`Restart`->`Console`中输入`run xxus/ps`->`Run All`。假如缺少命令行输入，波形将无限跑下去，直到`Break`
  - 命令行操作：`mips.tcl`

```
run xxus/ps
exit
```

1.CPU设计综述

1.1.信号

1.1.1.命名

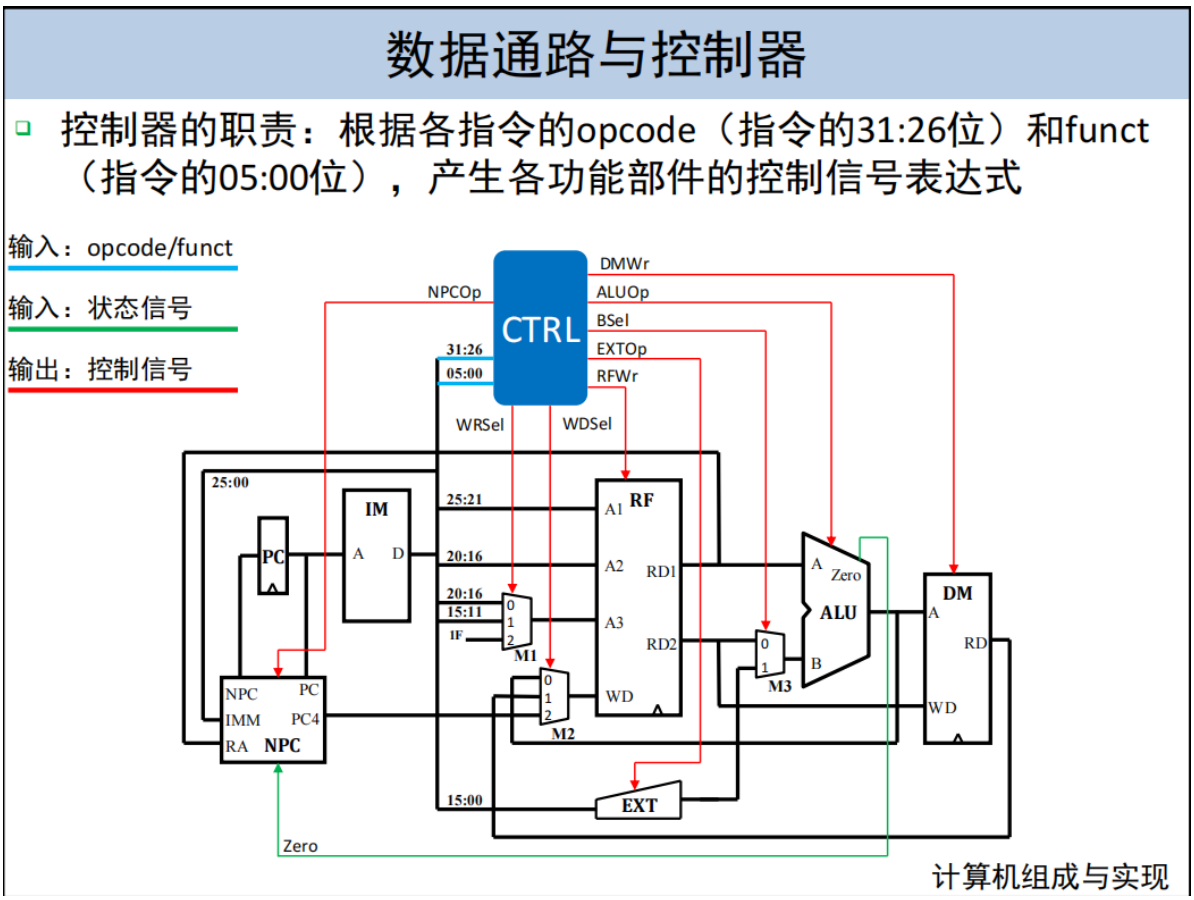
- 端口：命名与Logisim电路保持一致
- 控制：`xxOp_yy`
- 多路选择`xxSel_yy`
- 使能`电路名功能En`

1.2.模块

1.2.1.顶层

架构

参考高小鹏老师的PPT



端口说明

信号名称	描述
clk	时钟信号
reset	同步复位

1.2.2.IFU

端口说明

信号名称	方向	描述
clk	I	时钟信号
reset	I	同步复位
NPC[31:0]	I	NPC
Instr[31:0]	O	指令机器码
PC[31:0]	O	PC

1.2.3.CU

端口说明

信号名称	方向	描述
opcode[5:0]	I	指令机器码[31:26] - opcode部分
func[5:0]	I	指令机器码[5:0] - func部分
NPCOp[3:0]	O	NPC的控制信号
RFWrEn	O	RF的写使能
EXTOp	O	EXT的控制信号
ALUOp[3:0]	O	ALU的控制信号
ALUASel[1:0]	O	ALU-A的选择信号
ALUBSel[1:0]	O	ALU-B的选择信号
DMWrEn	O	DM的写使能
RFWRSel[2:0]	O	RF的写寄存器编号
RFWDSel[2:0]	O	RF的写数据选择
CMPOp[3:0]	O	ALU-CMP-B指令的控制信号
DMOp[1:0]	O	DM的控制信号
DMEXTOp	O	DM的符号拓展选择信号

真值表

	NPCOp[3:0]	RFWrEn	RFWRSel[2:0]	RFWDSel[2:0]	EXTOp	ALUOp[3:0]	CMPOp[3:0]	ALUASel[1:0]	ALUBSel[1:0]	DMWrEn	DMOp[1:0]	DMEXTOp
add	NPCOp_PC4	1	RFWRSel_rd	RFWDSel_ALU		ADD		ALUASel_rs	ALUBSel_rt			
sub	NPCOp_PC4	1	RFWRSel_rd	RFWDSel_ALU		SUB		ALUASel_rs	ALUBSel_rt			
ori	NPCOp_PC4	1	RFWRSel_rt	RFWDSel_ALU	EXTOp_zero	OR		ALUASel_rs	ALUBSel_imm			
lui	NPCOp_PC4	1	RFWRSel_rt	RFWDSel_ALU		LUI		ALUASel_rs	ALUBSel_imm			
lw	NPCOp_PC4	1	RFWRSel_rt	RFWDSel_DMRD	EXTOp_sign	ADD		ALUASel_rs	ALUBSel_imm		DMOp_w	
sw	NPCOp_PC4	0			EXTOp_sign	ADD		ALUASel_rs	ALUBSel_imm	1	DMOp_w	
jal	NPCOp_JAL	1	RFWRSel_31	RFWDSel_PC4				ALUASel_rs				
jr	NPCOp_JR	0						ALUASel_rs				
beq	NPCOp_Br	0					CMPOp_beq	ALUASel_rs	ALUBSel_rt			

1.2.4.DM

端口说明

信号名称	方向	描述
clk	I	时钟信号
reset	I	同步复位
Instr[31:0]	I	指令
PC[31:0]	I	PC
DMWrEn	I	DM写使能
DMMemAddr[31:0]	I	DM写入地址
DMRawData[31:0]	I	DM生数据
DMEXTOp	I	DM扩展控制信号
DMOp[1:0]	I	DM控制信号
DMRD[31:0]	O	DM读出数据

控制信号

DMOp[1:0]	描述
`DMOp_w`	以字为单位读/写
`DMOp_h`	以半字为单位读/写
`DMOp_b`	以字节为单位读/写

DMEXTOp	描述
`DMEXTOp_zero`	零扩展
`DMEXTOp_sign`	符号扩展

1.2.5.EXT

端口说明

信号名称	方向	描述
IMM16[15:0]	I	16位立即数
EXTOp	I	扩展控制信号
IMMEXT[31:0]	O	32位扩展后的立即数

1.2.6.NPC

端口说明

信号名称	方向	描述
PC[31:0]	I	PC
NPCOp[3:0]	I	NPC的控制信号
IMM26[25:0]	I	26位立即数
IMM16[15:0]	I	16位立即数
RA[31:0]	I	rs(ra)存的数据
Branch	I	B指令跳转的选择信号
PC4[31:0]	O	PC+4
NPC[31:0]	O	NPC

控制信号

NPCOp	描述
`NPCOp_PC4`	顺序执行
`NPCOp_Br`	B指令
`NPCOp_JAL`	JAL指令
`NPCOp_JR`	JR指令

选择信号

Branch	描述
0	顺序
1	跳转

1.2.7.RF

端口说明

信号名称	方向	描述
clk	I	时钟信号
reset	I	同步复位
A1[4:0]	I	rs寄存器编号
A2[4:0]	I	rt寄存器编号
A3[4:0]	I	写寄存器编号
WD[31:0]	I	写数据
RFWrEn	I	写使能
PC[31:0]	I	PC
RD1[31:0]	O	GRF[rs]
RD2[31:0]	O	GRF[rt]

选择信号

RFWRSel[2:0]	描述
`RFWRSel_rd`	写寄存器：rd
`RFWRSel_rt`	写寄存器：rt
`RFWRSel_31`	写寄存器：$31

RFWDSel[2:0]	描述
`RFWDSel_ALU`	写数据来自ALU
`RFWDSel_DMRD`	写数据来自DM读出的数据
`RFWDSel_PC4`	写数据=PC+4

1.2.8.ALU

端口说明

信号名称	方向	描述
A[31:0]	I	操作数A
B[31:0]	I	操作数B
ALUOp[3:0]	I	ALU的选择信号
CMPOp[3:0]	I	处理B指令的选择信号
Branch	O	B指令跳转的选择信号
ALURes[31:0]	O	ALU计算结果

控制信号

ALUOp[3:0]	描述
`ADD`	加法
`SUB`	减法
`OR`	或
`LUI`	立即数置高

CMPOp[3:0]	描述
`CMPOp_beq`	beq指令

选择信号

ALUASel	描述
`ALUASel_rs`	rs作A
`ALUASel_sh`	shamt作A

ALUBSel	描述
`ALUBSel_rt`	rt作B
`ALUBSel_imm`	IMMEXT作B

2.测试方案

> 在P3的基础上，实现更进一步的自动化和指令拓展

2.1.自动生成asm

2.1.1.覆盖范围

- `add`,`sub`
  - 注意：按**无符号**加减法处理。因此若用样例跑MARS会出现`Arithmetic Overflow`的报错提示。
- `ori`, `lw`, `sw`,`lui`,`beq`
- `jal`,`jr`
-

2.1.2.取值

任一样例中出现边界值的概率接近百分百。

- 寄存器
  - 0附近
  - 32位数边界附近

    - 32位数范围内的随机数（为保证lw和sw过程时地址不超过ROM和RAM的范围，故设置得较小）
- 无符号16位立即数（`ori`,`lui`）
    - 0附近
    - 16位无符号数边界附近
    - 随机数

2.1.3.跳转方向

- 向后跳
- 向前跳

  beq不参与其他指令的随机生成程序，自成一个模块，以特定的频率穿插在代码中。

  设置独立的跳转区，避免死循环。

2.2.Mars命令行

2.2.1.机器码

```
os.system("java -jar "+ test_path +"\\Mars_CO_v0.4.1.jar
"+record_path+"\\test.asm nc mc CompactDataAtZero a dump .text HexText
"+cpu_path+"\\code.txt")
```

2.2.2.标答

> 使用学长Toby Shi的魔改Mars生成标答

```
os.system("java -jar "+ test_path +"\\Mars_CO_v0.4.1.jar
"+record_path+"\\test.asm mc CompactLargeText coL1 cl bhelbal.class ig nc > "
+ res_path + "\\mars.txt")
```

2.3.ISE命令行

编写prj文件和tcl文件

tcl文件内容如下

```
run 100us
exit
```

ISE命令行：

```
os.system("D:\\BUAA\\software\\ISE\\Xilinx\\14.7\\ISE_DS\\ISE\\bin\\nt64\\fus
e -nodebug -prj "+ cpu_path +"\\mips.prj -o mips.exe mips_tb > compile.txt")
```

2.4.无人值守测试

将上述步骤封装成为函数，使用循环进行批量测试。将必要文件和日志存储，方便后期查询。

3.思考题

1. 阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0]？

因为DM以字为一个单位，但MIPS以字节为单位，addr最低两位一定是0。所以应当取[11:2]

2. 思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。

   指令对应的控制信号如何取值：便于验证指令行为的正确性。在针对单条指令进行调试时很有好处。但多种控制信号混杂，无法关注其他指令，没有整体性。

   控制信号每种取值所对应的指令：便于总揽全局，宏观部署控制信号的结构。但是难以针对单条指令进行调试，缺乏局部性。

3. 在相应的部件中，复位信号的设计都是**同步复位**，这与 P3 中的设计要求不同。请对比**同步复位**与**异步复位**这两种方式的 reset 信号与 clk 信号优先级的关系。

   同步复位：clk优先。在clk位于上升沿的时候，考虑reset的取值。

   异步复位：reset和clk优先级相同。

4. C 语言是一种弱类型程序设计语言。C 语言中不对计算结果溢出进行处理，这意味着 C 语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持 C 语言，MIPS 指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的 Operation 部分。

   根据 RTL 语言描述： addi 与 addiu 的区别在于当出现溢出时， addiu 忽略溢出，并将溢出的最高位舍弃； addi 会报告 SignalException(IntegerOverflow) 。故忽略溢出时，二者等价。